Networks Project

Instructor:

Dr. Moosavi

Prepared by:

Hesameddin Fathi

Student Number:

40330795

Fall 2024

Table of Contents

۱–صورت پروژه

پیش بینی تعداد خودرو یا سرعت خودروها در عضوی از قسمت های گراف بر اساس دنباله زمانی تعداد یا سرعت خودروها در گذشته و یا شرایط خیابان های اطراف در همان لحظه یا در گذشته.

بنابراین ورودی الگوریتم مجموعه ای از دنباله های زمانی و خروجی نیز یک دنباله زمانی از اعداد که باید با داده های واقعی همان بازه زمانی مقایسه شود.

ابتدا مقاله ای که بر اساس آن پروژه را پیاده سازی کرده ایم را بررسی می کنیم.

۲–مقاله

۲–۱–عنوان و هدف

مقاله به بررسی مدلی جدید برای پیش‌بینی جریان ترافیک می‌پردازد و هدف اصلی، طراحی یک مدل است که بتواند اطلاعات مکانی (مانند موقعیت جاده‌ها و تقاطع‌ها) و اطلاعات زمانی (الگوهای تغییرات جریان ترافیک در طول زمان) را به‌طور همزمان مدلسازی کند تا دقت پیش‌بینی افزایش یابد.

۲–۲–چالش های موجود در پیش بینی جریان ترافیک

- **وابستگی‌های پیچیده:** جاده‌ها و حسگرهای ترافیکی به دلیل فاصله و ویژگی‌های متفاوت، ارتباطات پیچیده‌ای دارند.
- **الگوهای زمانی متغیر:** جریان ترافیک در زمان‌های مختلف (مثلا اوج ترافیک در ساعات شلوغی) تغییر می‌کند.
- **محدودیت گراف‌های مکانی ثابت:** بسیاری از مدل‌های پیشین تنها از گراف‌های مکانی از پیش تعیین‌شده استفاده می‌کردند و به وابستگی‌های زمانی کمتر توجه داشتند.

۲–۳–ایده اصلی روش پیشنهادی(STFGNN)

ساخت گراف زمانی داده محور: به جای تکیه بر گراف‌های مکانی از پیش تعیین‌شده، نویسندگان یک روش داده‌محور برای ساخت «گراف زمانی» ارائه می‌دهند.که این گراف با استفاده از الگوریتم هایی مانند Dynamic Time Warping (DTW) و نسخه سریع آن fast-DTW ایجاد می شود تا شباهت بین سری های زمانی (مثلا تغییرات ترافیکی در دو نقطه مختلف) را محاسبه کند. چالشی که در در ساخت گراف زمانی وجود داشت این بود که برای اندازه گیری شباهت زمانی بین نقاط از الگوریتم DTM استفاده می شود و مشکل این الگوریتم این است که پیچیدگی محاسباتی بالایی دارد و از مرتبه $O(n^2)$ می باشد و برای مجموعه داده های بزرگ بسیار کند است که برای بهینه سازی نسخه ای سریع تر به نام fast-DTW که پیچیدگی را کاهش داده و آن را قابل اجرا برای داده های بزرگ می کند، در این روش در واقع محدوده جستجوی شباهت بین داده های زمانی محدود می شود که باعث کاهش زمان محاسباتی می شود.

ترکیب گراف های مکانی و زمانی: گراف مکانی که از داده های مکانی استخراج می شود و گراف زمانی تولید شده، با یکدیگر ترکیب می شوند تا یک گراف مکانی-زمانی ساخته شود که این گراف ترکیبی قادر است وابستگی‌های بین حسگرها از دو منظر مکانی و زمانی را به‌طور همزمان در نظر بگیرد.

۲-۴-نمایش شبکه جاده ها به عنوان گراف

شبکه جاده ای به عنوان یک گراف (V , E , A_{SG}) نمایش داده می شود که V مجموعه ای از گره ها (مثل حسگرها یا جاده ها)، E مجموعه ای از یال ها که ارتباط بین گره ها را نشان می دهد و A_{SG} ماتریس مجاورت مکانی که نشان می دهد کدام حسگرها یا جاده ها به هم نزدیک هستند و چه فاصله ای دارند.

۲-۵-بخش های اصلی مدل

لایه ورودی: داده‌های ترافیکی (مثل سرعت و تعداد خودروها) به این لایه داده می‌شود و از تابع فعال‌سازی ReLUبرای پردازش داده‌ها استفاده می‌کند.

لایه‌های پردازش مکانی-زمانی: شامل چندین ماژول STFGN است که به‌طور موازی اجرا می‌شوند و یک ماژول CNNگیت‌دار (Gated CNN Module)دارد که شامل دو کانولوشن ۱بعدی Dilated است یعنی هر کدوم از کانولوشن‌ها، یک سری الگوهای خاص رو در داده‌ها پیدا می‌کنند و بعد این اطلاعات با هم ترکیب می‌شن گیت هم مثل یه دروازه عمل می‌کند که به مدل اجازه می‌دهد فقط اطلاعات مهم و مرتبط رو از هر کانولوشن انتخاب کنه که هدف این بخش ترکیب وابستگی‌های مکانی و زمانی برای بهبود دقت پیش‌بینی است.

لایه خروجی: شامل دو لایه کاملا متصل است که نتیجه نهایی را تولید می‌کنند که این لایه نیز از تابع فعال‌سازی ReLU استفاده می‌کند.

۲-۶-نحوه ساخت گراف مکانی-زمانی ترکیبی

مدل نهایی از سه نوع گراف زیر استفاده می کند و در نهایت یک گراف ترکیبی مکانی-زمانی A_{STFG} را تشکیل می دهند.

۱- **گراف مکانی A_{SG}:** بر اساس فاصله فیزیکی بین جاده ها

۲- **گراف زمانی A_{TG}:** پیدا کردن ارتباط بین گره‌هایی که رفتار زمانی مشابهی دارند، حتی اگر از نظر مکانی دور از هم باشند و این گراف بر اساس شباهت بین الگوهای زمانی گره‌ها ساخته می‌شود مثلا دو خیابان در دو منطقه متفاوت از شهر ممکن است در ساعات اوج (مثلا ۸ صبح) هم‌زمان ترافیک سنگینی داشته باشند که مدل با بررسی این الگوها، این دو گره را در گراف زمانی بهم متصل می کند.

۳- **گراف اتصال زمانی A_{TC}:** هدف نشان دادن ارتباط هر گره با خودش در لحظات مختلف زمان است در واقع این گراف ارتباط هر گره را با خودش در زمان‌های قبل و بعد نشان می‌دهد و این ساختار کمک می‌کند تا اطلاعات وضعیت یک گره در طول زمان حفظ شود مثلا اگر یک خیابان در ساعت ۸ صبح دارای ترافیک سنگین باشد، احتمال زیادی وجود دارد که در ساعت ۸:۱۰ دقیقه نیز همچنان شلوغ باشد.

۲–۷–نحوه پردازش اطلاعات در ماژول STFGN

در اینجا توضیح می‌دهیم که چگونه ماژول STFGN می‌تواند اطلاعات مکانی-زمانی را پردازش کرده و روابط پیچیده را استخراج کند. این ماژول یکی از اجزای کلیدی مدل STFGNN است که وابستگی‌های مکانی-زمانی را یاد می‌گیرد.

ورودی: داده‌های مربوط به جاده‌ها (یا حسگرها) که شامل اطلاعات مکانی و زمانی است.

استفاده از گراف ترکیبی A_{STFG} این ماژول با استفاده از ماتریس A_{STFG} ارتباط بین گره ها را پردازش می کند.

روش پردازش: به جای روش‌های پیچیده در شبکه‌های عصبی گرافی(مثل استفاده از فیلتر لاپلاس در GCN)، این ماژول فقط چندین ضرب ماتریسی ساده انجام می دهد که سریع تر و کم هزینه تر است.

۲–۸–مکانیزم های یادگیری در ماژول STFGN

استفاده از مکانیزم گیت دار (Gating Mechanism): در این ماژول از واحدهای خطی گیت دار(GLU) استفاده شده است که مشابه LSTM/RNN عمل می کنند و باعث می شود مدل بتواند الگوهای غیرخطی را یاد بگیرد و این ویژگی به مدل کمک می کند که فقط اطلاعات مهم را عبور دهد و از ورود نویزهای اضافی جلوگیری کند.

۲–۹–پردازش چندلایه ای و ترکیب داده ها

افزودن ارتباطات پسماندی Residual Connections:

- این تکنیک کمک می‌کند که اطلاعات گم نشود و مدل وابستگی‌های طولانی‌مدت را بهتر یاد بگیرد.

استفاده ازMaxPooling :

- برای کاهش ابعاد و تمرکز روی ویژگی‌های مهم‌تر، از عملگر MaxPooling روی تمام حالات مخفی Hidden States استفاده می‌شود.

برش و انتخاب ویژگی‌های مهم:

- فقط بخش مرکزی داده‌های پردازش‌شده برای مراحل بعدی نگه داشته می‌شود تا از پیچیدگی غیرضروری جلوگیری شود.

پردازش موازی:

- چندین ماژول STFGN به‌طور مستقل و موازی داده‌ها را پردازش می‌کنند که باعث صرفه‌جویی در زمان محاسباتی می‌شود.

ادغام با CNN گیت‌دار:

- خروجی تمام ماژول‌های STFGN با خروجی Gated CNN ترکیب شده و به عنوان ورودی به لایه بعدی داده می‌شود.

۳– پیاده سازی پروژه

۳–۱–ساخت ماتریس مجاورت فضایی

برای ساخت ماتریس Spatial Adjacency Matrix باید از فایل PeMS08_rel که داده های مکانی در آن قرار دارد استفاده کنیم، که شامل ۵ ستون است ولی ما فقط به سه ستون آن احتیاج داریم. بنابراین با کمک کد زیر سه ستون مورد نیاز را استخراج می کنیم و در فایل جدید سیو می کنیم.

```python
import pandas as pd

# Load the Excel file
df = pd.read_csv("PeMS08_rel.csv")

# Keep only the desired columns
df = df[['origin_id', 'destination_id', 'cost']]

# Save the updated DataFrame to a new CSV file
df.to_csv('matris_mojaverat_fazayi.csv', index=False)
```

۳–۲–ساخت ماتریس مجاورت زمانی

قبل از ایجاد ماتریس مجاورت زمانی لازم است تا دیتاست خود را آماده کنیم و داده های خود را به صورت آرایه Numpy ذخیره کنیم. در این مرحله همچنین ستون های اضافی را حذف می کنیم و فقط ستون های مورد نیاز را نگه می داریم.

که همانطور که مشخص است فقط ستون های traffic_flow,traffic_ocupancy,traffic_speed را نگه می داریم و آنها را به یک آرایه numpy تبدیل می کنیم و در آخر آن را در یک فایل npz ذخیره می کنیم.

```python
import numpy as np
import pandas as pd

file_path = "PeMS08.csv"

print("data is loading..")
df = pd.read_csv(file_path, sep=",")

df["time"] = pd.to_datetime(df["time"])
df = df.sort_values(by="time")

features = ["traffic_flow", "traffic_occupancy", "traffic_speed"]

df[features] = df[features].apply(lambda x: x.fillna(x.mean()), axis=0)

unique_sensors = sorted(df["entity_id"].unique())
time_steps = df["time"].nunique()

sensor_to_index = {sensor: i for i, sensor in enumerate(unique_sensors)}

data_array = np.zeros((time_steps, len(unique_sensors), len(features)), dtype=np.float32)

for i, (timestamp, group) in enumerate(df.groupby("time")):
    for _, row in group.iterrows():
        sensor_idx = sensor_to_index[row["entity_id"]]
        data_array[i, sensor_idx, :] = row[features].values

np.savez_compressed("PEMS08.npz", data=data_array)
print("npz file created")
```

ماتریس Temporal Adjacency Matrix را با کمک کد زیر که در پایتون نوشتیم ایجاد می کنیم.

این کد فاصله‌های Dynamic Time Warping (DTW) را بین سری‌های زمانی گره‌های مختلف در یک مجموعه داده محاسبه می‌کند و بر اساس این فاصله‌ها یک گراف زمانی می‌سازد. این گراف روابط بین گره‌های مختلف را بر اساس شباهت الگوهای آنها در طول زمان نشان می‌دهد.

توضیح بخش های مختلف کد در زیر آورده شده است.

```
def normalize(a):
    mu=np.mean(a,axis=1,keepdims=True)
    std=np.std(a,axis=1,keepdims=True)
    return (a-mu)/std
```

این تابع مقدار میانگین (mu) و انحراف معیار (std) را برای هر سری زمانی محاسبه می‌کند.

نتیجه یک آرایه نرمال شده است که در آن مقادیر به گونه‌ای تبدیل شده‌اند که میانگین صفر و واریانس واحد داشته باشند.

```
def compute_dtw(a,b,order=1,Ts=12,normal=True):
    if normal:
        a=normalize(a)
        b=normalize(b)
    T0=a.shape[1]
    d=np.reshape(a,[-1,1,T0])-np.reshape(b,[-1,T0,1])
    d=np.linalg.norm(d,axis=0,ord=order)
    D=np.zeros([T0,T0])
    for i in range(T0):
        for j in range(max(0,i-Ts),min(T0,i+Ts+1)):
            if (i==0) and (j==0):
                D[i,j]=d[i,j]**order
                continue
            if (i==0):
                D[i,j]=d[i,j]**order+D[i,j-1]
                continue
            if (j==0):
                D[i,j]=d[i,j]**order+D[i-1,j]
                continue
            if (j==i-Ts):
                D[i,j]=d[i,j]**order+min(D[i-1,j-1],D[i-1,j])
                continue
            if (j==i+Ts):
                D[i,j]=d[i,j]**order+min(D[i-1,j-1],D[i,j-1])
                continue
            D[i,j]=d[i,j]**order+min(D[i-1,j-1],D[i-1,j],D[i,j-1])
    return D[-1,-1]**(1.0/order)
```

در اینجا تابع compute dtw فاصله Dynamic Time Warping را بین دو نقطه a و b محاسبه می کند و اگر normal=True باشد، ابتدا داده‌ها را نرمال‌سازی می‌کند، سپس اختلاف مقدار هر نقطه از سری a و b را محاسبه کرده و آن را در ماتریس d ذخیره می‌کند.

در حلقه for ، الگوریتم DTW پیاده‌سازی شده و مقدار نهایی فاصله‌ی DTW بین دو سری زمانی برگردانده می‌شود.

```
df = np.load("PeMS08.npz")['data']
num_samples,ndim,_ = df.shape
num_train = int(num_samples * 0.6)
num_dtw=int(num_train/args.period)*args.period
data=df[:num_dtw,:,:1].reshape([-1,args.period,ndim])
```

داده‌های npz بارگذاری شده و num_samples (تعداد نمونه‌ها) و ndim (تعداد حسگرها) استخراج می‌شود.

num_train مقدار ۶۰٪ از داده‌ها را مشخص می‌کند (برای آموزش استفاده می‌شود).

داده‌ها برای تطبیق با بازه‌های زمانی period تنظیم می‌شوند.

```
d=np.zeros([ndim,ndim])

for i in range(ndim):
    t1=time.time()
    for j in range(i+1,ndim):
        d[i,j]=compute_dtw(data[:,:,i],data[:,:,j],order=args.order,Ts=args.lag)
    t2=time.time()
    print('Line',i,'finished in',t2-t1,'seconds.')
```

این قسمت، ماتریس فاصله زمانی DTW را برای همه جفت حسگرها محاسبه می‌کند.

نتایج در یک ماتریس d ذخیره می‌شود.

```
dtw=d+d.T

np.save("./newdataset/"+args.dataset+"-dtw-"+str(args.period)+'-'+str(args.order)+"-.npy",dtw)
print("The calculation is done!")
```

ماتریس DTW ایجاد شده و در مسیر newdataset ذخیره می‌شود.

```
adj = np.load("./newdataset/"+args.dataset+"-dtw-"+str(args.period)+'-'+str(args.order)+"-.npy")
adj = adj+ adj.T
w_adj = np.zeros([ndim,ndim])
adj_percent = args.sparsity
top = int(ndim * adj_percent)
```

گراف زمانی ایجاد می‌شود، که فقط top درصد از نزدیک‌ترین ارتباطات را حفظ می‌کند.

```
for i in range(adj.shape[0]):
    a = adj[i,:].argsort()[0:top]
    for j in range(top):
        w_adj[i, a[j]] = 1
```

این قسمت تنها top% از اتصالات را فعال نگه می‌دارد.

```
for i in range(ndim):
    for j in range(ndim):
        if (w_adj[i][j] != w_adj[j][i] and w_adj[i][j] ==0):
            w_adj[i][j] = 1
        if( i==j):
            w_adj[i][j] = 1
```

گراف نهایی قرینه سازی می‌شود اگر (i, j) مقدار داشته باشد، (j, i) هم مقدار خواهد داشت.

و در آخر ماتریس نهایی گراف زمانی (adj_tg_PEMS08.csv) ذخیره می‌شود.

۳-۳-تنظیمات مدل

در این فایل پیکربندی مدل STFGNN را ذخیره می کنیم و هر مقدار در این فایل یک پارامتر کلیدی برای اجرای مدل روی دیتاست PEMS08 می باشد.

```
{
    "module_type": "individual",
    "act_type": "GLU",
    "temporal_emb": true,
    "spatial_emb": true,
    "use_mask": true,
    "first_layer_embedding_size": 64,
    "filters": [
        [64, 64, 64],
        [64, 64, 64],
        [64, 64, 64]
    ],
    "batch_size": 32,
    "optimizer": "adam",
    "learning_rate": 1e-3,
    "epochs": 2,
    "max_update_factor": 1,
    "ctx": -1,
    "adj_filename": "matris_mojaverat_fazayi.csv",
    "id_filename": null,
    "graph_signal_matrix_filename": "PEMS08new.npz",
    "num_of_vertices": 170,
    "points_per_hour": 12,
    "num_for_predict": 12,
    "num_of_features": 1,
    "adj_dtw_filename": "adj_PEMS08_001_new.csv"
}
```

مدل از نوع "individual"است، یعنی هر گره (حسگر) به صورت مجزا در مدل پردازش می‌شود.

GLU (Gated Linear Unit) برای بهبود یادگیری روابط غیرخطی در داده‌های زمانی استفاده می‌شود.

temporal_emb مشخص می‌کند که مدل ویژگی‌های زمانی را در نظر بگیرد.

spatial_emb مشخص می‌کند که مدل ویژگی‌های مکانی (ارتباط بین حسگرها) را در نظر بگیرد.

استفاده از ماسک کردن داده‌ها، مثلاً برای نادیده گرفتن مقادیر گمشده در ورودی‌ها

تعداد نورون‌ها در لایه ورودی را تعیین می‌کند بعد تعبیه اولیه Embedding Size

مشخص می‌کند که ۳ لایه‌ی کانولوشنی (CNN) در مدل وجود دارد.

هر لایه دارای ۳ فیلتر با اندازه‌ی ۶۴ است.

این فیلترها روابط مکانی-زمانی بین داده‌های حسگرها را استخراج می‌کنند.

بچ سایز اندازه‌ی کمترین بچ در طول آموزش را مشخص می‌کند. (مدل در هر مرحله ۳۲ نمونه داده را پردازش می‌کند)

الگوریتم بهینه‌سازی مدل Adam است، که یکی از سریع‌ترین و پایدارترین روش‌ها برای بهینه‌سازی شبکه‌های عصبی است.

نرخ یادگیری (Learning Rate) مدل، مقدار 0.001 در نظر گرفته شده است.

تعداد دوره‌های آموزشی (Epochs) را مشخص می‌کند، یعنی مدل ۲ بار کل داده‌ها را پردازش می‌کند.

0=Ctx مشخص می‌کند که مدل روی کارت گرافیک GPU شماره ۰ اجرا شود و اگر مقدار 1- باشد، مدل روی CPU اجرا خواهد شد.

در آخر هم مسیر فایل های داده را مشخص می کنیم:

Adj_filename: مسیر فایل ماتریس مجاورت مکانی (Spatial Adjacency Matrix)

graph_signal_matrix_filename: مسیر فایل ماتریس سیگنال‌های گراف که شامل داده‌های جریان ترافیک، سرعت و تراکم ترافیکی در طول زمان است.

تعداد گره‌ها (Sensors) در گراف برابر با ۱۷۰ حسگر ترافیکی است.

تعداد بازه‌های زمانی در هر ساعت چون داده‌ها هر ۵ دقیقه ثبت شده‌اند، در یک ساعت ۱۲ نمونه داریم.

مدل قرار است ۱۲ بازه‌ی زمانی آینده را پیش‌بینی کند یعنی مدل ترافیک ۱ ساعت آینده را پیش‌بینی می‌کند.

adj_dtw_filename: مسیر فایل ماتریس مجاورت زمانی (Temporal Adjacency Matrix) که با الگوریتم DTW (Dynamic Time Warping) محاسبه شده است و در مرحله قبل بدستش آوردیم.

۳-۴-تعریف توابع برای لایه های گراف عصبی

کد زیر یکی از بخش های اصلی مدل STFGNN است که برای پیش بینی ترافیک استفاده می شود. این کد شامل تعریف توابعی برای لایه‌های شبکه گراف عصبی (Graph Neural Network) و پردازش داده‌های ترافیکی است.

```
def position_embedding(data,
                       input_length, num_of_vertices,embedding_size,
                       temporal=True,spatial=True,
                       init=mx.init.Xavier(magnitude=0.0003), prefix=""):
```

position_embedding اطلاعات موقعیتی (زمانی و یا مکانی) را به داده‌های ورودی اضافه می‌کند.

به طور خلاصه، این تابع داده‌های ورودی (data) را به همراه اطلاعات مربوط به ابعاد آن input_length, num_of_vertices embedding_size، نوع جاسازی مورد نیاز (temporal, spatial) ، نحوه مقداردهی اولیه (init) و یک پیشوند اختیاری (prefix) دریافت می‌کند.

Init نحوه مقداردهی اولیه وزن‌های بردارهای جاسازی را مشخص می‌کند .مقدار پیش‌فرض از مقداردهی اولیه Xavier استفاده می‌کند که یک روش رایج برای مقداردهی اولیه وزن‌ها در شبکه‌های عصبی است magnitude .دامنه مقادیر تصادفی را کنترل می‌کند.

```
if temporal:
    # shape is (1, T, 1, C)
    temporal_emb = mx.sym.var(
        "{}_t_emb".format(prefix),
        shape=(1, input_length, 1, embedding_size),
        init=init
    )
```

اگر temporal=True باشد، ماتریس جاسازی زمانی با ابعاد (1, T, 1, C) ساخته می‌شود.

```
if spatial:
    # shape is (1, 1, N, C)
    spatial_emb = mx.sym.var(
        "{}_v_emb".format(prefix),
        shape=(1, 1, num_of_vertices, embedding_size),
        init=init
    )
```

اگر spatial=True باشد، ماتریس جاسازی مکانی با ابعاد (1, 1, N, C) ساخته می‌شود.

```
if temporal_emb is not None:
    data = mx.sym.broadcast_add(data, temporal_emb)
if spatial_emb is not None:
    data = mx.sym.broadcast_add(data, spatial_emb)
```

این دو خط کد، جاسازی‌های زمانی و مکانی ایجاد شده را به داده‌های ورودی (data) اضافه می‌کنند.

```python
def gcn_operation(data, adj, num_of_filter, num_of_features, num_of_vertices,
                  activation, prefix=""):

    assert activation in {'GLU', 'relu'}
    # shape is (4N, B, C)
    data = mx.sym.dot(adj, data)

    if activation == 'GLU':
        data = mx.sym.FullyConnected(
            data, flatten=False, num_hidden=2 * num_of_filter
        )
        lhs, rhs = mx.sym.split(data, num_outputs=2, axis=2)
        return lhs * mx.sym.sigmoid(rhs)

    elif activation == 'relu':
        return mx.sym.Activation(
            mx.sym.FullyConnected(data, flatten=False, num_hidden=num_of_filter),
            activation
        )
```

این تابع gcn_operation یک لایه‌ی GCN پایه را پیاده‌سازی می‌کند .ابتدا اطلاعات را از همسایه‌ها با ضرب ماتریس مجاورت جمع‌آوری می‌کند، سپس یک تبدیل خطی با FullyConnected و یک تابع فعال‌سازی غیرخطی GLU یا relu را اعمال می‌کند تا ویژگی‌های جدیدی برای هر گره ایجاد کند، این ویژگی‌های جدید، ترکیبی از ویژگی‌های قبلی گره و اطلاعات جمع‌آوری‌شده از همسایگانش هستند.

اگر GLU باشد، خروجی را به دو بخش تقسیم کرده و با سیگموید ترکیب می‌کند.

اگر ReLU باشد، یک لایه FullyConnected و تابع فعال‌سازی ReLU اعمال می‌شود.

```python
def stsgcm(data, adj, filters, num_of_features, num_of_vertices, activation, prefix=""):

    need_concat = []
    for i in range(len(filters)):
        data = gcn_operation(
            data, adj, filters[i], num_of_features, num_of_vertices,
            activation=activation, prefix="{}_gcn_{}".format(prefix, i)
        )
        need_concat.append(data)
        num_of_features = filters[i]

    need_concat = [
        mx.sym.expand_dims(
            mx.sym.slice(
                i,
                begin=(num_of_vertices, None, None),
                end=(2 * num_of_vertices, None, None)
            ), 0
        ) for i in need_concat
    ]
    return mx.sym.max(mx.sym.concat(*need_concat, dim=0), axis=0)
```

تابع stsgcm چندین لایه GCN را به صورت پشته‌ای اجرا می‌کند، سپس اطلاعات مربوط به گره‌های اصلی را از خروجی هر لایه استخراج می‌کند، و در نهایت با استفاده از عملیات بیشینه‌سازی، مهم‌ترین ویژگی‌ها را از بین تمام لایه‌ها انتخاب می‌کند. این تابع یک ماژول کانولوشن گراف جدا شده‌ی مکانی-زمانی (spatio-temporal separated graph convolutional module) را پیاده‌سازی می‌کند.

در حلقه در هر مرحله، یک لایه‌ی gcn_operation اضافه شده و ویژگی‌ها استخراج می‌شوند.

```
def stsgcl(data, adj, T, num_of_vertices, num_of_features, filters,
           module_type, activation, temporal_emb=True, spatial_emb=True, prefix=""):

    assert module_type in {'sharing', 'individual'}

    if module_type == 'individual':
        return sthgcn_layer_individual(
            data, adj, T, num_of_vertices, num_of_features, filters,
            activation, temporal_emb, spatial_emb, prefix
        )
```

تابع stsgcl یک لایه STSGCL را پیاده‌سازی می‌کند. این لایه می‌تواند از دو نوع ماژول تشکیل شده باشد sharing (که در این کد پیاده‌سازی نشده) و individual, در حالت individual برای هر گام زمانی یک ماژول STSGCM مجزا استفاده می‌شود که این کار با فراخوانی تابع sthgcn_layer_individual انجام می‌شود.

```
def sthgcn_layer_individual(data, adj,
                            T, num_of_vertices, num_of_features, filters,
                            activation, temporal_emb=True, spatial_emb=True,
                            prefix=""):

    data = position_embedding(data, T, num_of_vertices, num_of_features,
                             temporal_emb, spatial_emb,
                             prefix="{}_emb".format(prefix))

    data_temp = mx.sym.transpose(data, (0, 3, 2, 1))
    data_left = mx.sym.sigmoid(mx.sym.Convolution(data= data_temp, num_filter=num_of_features, kernel=(1, 2), stride=(1, 1), dilate=(1,3)))
    data_right = mx.sym.tanh(mx.sym.Convolution(data=data_temp, num_filter=num_of_features, kernel=(1, 2), stride=(1, 1), dilate=(1,3)))
    data_time_axis = data_left * data_right
    data_res = mx.sym.transpose(data_time_axis, (0, 3, 2, 1))
    need_concat = []
    for i in range(T - 3):
        t = mx.sym.slice(data, begin=(None, i, None, None),
                         end=(None, i + 4, None, None))

        t = mx.sym.reshape(t, (-1, 4 * num_of_vertices, num_of_features))

        t = mx.sym.transpose(t, (1, 0, 2))

        t = stsgcm(
            t, adj, filters, num_of_features, num_of_vertices,
            activation=activation,
            prefix="{}_stsgcm_{}".format(prefix, i)
        )
        t = mx.sym.swapaxes(t, 0, 1)
        need_concat.append(mx.sym.expand_dims(t, axis=1))
    need_concat_ = mx.sym.concat(*need_concat, dim=1)
    layer_out = need_concat_ + data_res
    return layer_out
```

تابع sthgcn_layer_individual ابتدا یک پردازش اولیه روی محور زمان انجام می‌دهد (با استفاده از کانولوشن‌های یک‌بعدی). سپس، یک حلقه for اجرا می‌کند که در هر تکرار، یک برش ۴ تایی از داده‌ها می‌گیرد و یک ماژول STSGCM *مجزا* را روی آن اعمال می‌کند .در نهایت، خروجی‌های همه‌ی STSGCM ها را با هم ترکیب می‌کند و حاصل جمع را با خروجی پردازش اولیه جمع می کند و برمی‌گرداند .این تابع، پیاده‌سازی STSGCL با STSGCM های مستقل برای هر بخش از سری زمانی است.

```
def output_layer(data, num_of_vertices, input_length, num_of_features,
                 num_of_filters=128, predict_length=12):

    data = mx.sym.swapaxes(data, 1, 2)
    data = mx.sym.reshape(data, (-1, num_of_vertices, input_length * num_of_features))

    data = mx.sym.Activation(
        mx.sym.FullyConnected(
            data,
            flatten=False,
            num_hidden=num_of_filters
        ), 'relu'
    )
    data = mx.sym.FullyConnected(
    data, flatten=False, num_hidden=predict_length
    )

    data = mx.sym.swapaxes(data, 1, 2)
    return data
```

خروجی نهایی مدل و پیش‌بینی ترافیک

تابع output_layer داده‌ها را تغییر شکل می‌دهد، دو لایه کاملاً متصل با فعال‌سازی ReLU بین آن‌ها را اعمال می‌کند، و در نهایت دوباره داده‌ها را تغییر شکل می‌دهد تا پیش‌بینی‌هایی با شکل (B, T', N) تولید کند این تابع، یک لایه خروجی استاندارد برای مدل‌های پیش‌بینی سری زمانی (یا سری زمانی گرافی) است که در آن می‌خواهیم برای T' گام زمانی آینده، مقادیر را برای N گره پیش‌بینی کنیم.

```
def huber_loss(data, label, rho=1):

    loss = mx.sym.abs(data - label)
    loss = mx.sym.where(loss > rho, loss - 0.5 * rho,
                        (0.5 / rho) * mx.sym.square(loss))
    loss = mx.sym.MakeLoss(loss)
    return loss
```

تابع huber_loss یک تابع زیان مقاوم در برابر داده‌های پرت را پیاده‌سازی می‌کند. این تابع برای خطاهای کوچک شبیه MSE و برای خطاهای بزرگ شبیه MAE عمل می‌کند .پارامتر rho نقطه‌ی گذار بین این دو رفتار را کنترل می‌کند .این تابع زیان در مسائل رگرسیون، به ویژه زمانی که داده‌ها ممکن است شامل داده‌های پرت باشند، مفید است.

```python
def weighted_loss(data, label, input_length, rho=1):

    weight = mx.sym.expand_dims(
        mx.sym.expand_dims(
            mx.sym.flip(mx.sym.arange(1, input_length + 1), axis=0),
            axis=0
        ), axis=-1
    )
    agg_loss = mx.sym.broadcast_mul(
        huber_loss(data, label, rho),
        weight
    )
    return agg_loss
```

تابع weighted_loss یک وزن‌دهی به Huber Loss اعمال می‌کند .این وزن‌دهی به گونه‌ای است که به خطاهای مربوط به گام‌های زمانی اخیرتر در سری زمانی پیش‌بینی‌شده، وزن بیشتری می‌دهد. این کار با ایجاد یک تنسور وزن (که مقادیر آن به صورت نزولی از T' تا ۱ هستند) و ضرب آن در خروجی huber_loss انجام می‌شود .این تابع زیان، برای آموزش مدل‌هایی که باید روی پیش‌بینی دقیق‌تر آینده‌ی نزدیک تمرکز کنند، مناسب است.

```python
def stsgcn(data, adj, label,
           input_length, num_of_vertices, num_of_features,
           filter_list, module_type, activation,
           use_mask=True, mask_init_value=None,
           temporal_emb=True, spatial_emb=True,
           prefix="", rho=1, predict_length=12):

    if use_mask:
        if mask_init_value is None:
            raise ValueError("mask init value is None!")
        mask = mx.sym.var("{}_mask".format(prefix),
                          shape=(4 * num_of_vertices, 4 * num_of_vertices),
                          init=mask_init_value)
        adj = mask * adj

    for idx, filters in enumerate(filter_list):
        data = stsgcl(data, adj, input_length, num_of_vertices,
                      num_of_features, filters, module_type,
                      activation=activation,
                      temporal_emb=temporal_emb,
                      spatial_emb=spatial_emb,
                      prefix="{}_stsgcl_{}".format(prefix, idx))
        input_length -= 3
        num_of_features = filters[-1]

    need_concat = []
    for i in range(predict_length):
        need_concat.append(
            output_layer(
                data, num_of_vertices, input_length, num_of_features,
                num_of_filters=128, predict_length=1
            )
        )
    data = mx.sym.concat(*need_concat, dim=1)

    loss = huber_loss(data, label, rho=rho)
    return mx.sym.Group([loss, mx.sym.BlockGrad(data, name='pred')])
```

تابع stsgcn کل مدل STSGCN را پیاده‌سازی می‌کند. این تابع داده‌های ورودی، ماتریس مجاورت، برچسب‌ها و پارامترهای مختلف را دریافت می‌کند، یک ماسک (اختیاری) روی ماتریس مجاورت اعمال می‌کند، چندین لایه STSGCL را به صورت پشته‌ای اجرا می‌کند، یک لایه خروجی برای تولید پیش‌بینی‌ها اضافه می‌کند، Huber Loss را محاسبه می‌کند، و در نهایت، loss و پیش‌بینی‌ها را (با جلوگیری از محاسبه‌ی گرادیان برای پیش‌بینی‌ها) برمی‌گرداند. این تابع، یک مدل کامل و آماده‌ی آموزش برای پیش‌بینی سری‌های زمانی گرافی است.

این کد هسته‌ی اصلی مدل ما است که برای پیش‌بینی ترافیک جاده‌ای استفاده می‌شود.

۳-۵-توابع کمکی

```
def construct_model(config):
    from models.stsgcnn import stsgcn

    module_type = config['module_type']
    act_type = config['act_type']
    temporal_emb = config['temporal_emb']
    spatial_emb = config['spatial_emb']
    use_mask = config['use_mask']
    batch_size = config['batch_size']

    num_of_vertices = config['num_of_vertices']
    num_of_features = config['num_of_features']
    points_per_hour = config['points_per_hour']
    num_for_predict = config['num_for_predict']
    adj_filename = config['adj_filename']
    id_filename = config['id_filename']
```

این خطوط، مقادیر پارامترهای مختلف مدل را از دیکشنری config می‌خوانند .این پارامترها، جنبه‌های مختلف مدل (مانند نوع ماژول، تابع فعال‌سازی، استفاده از جاسازی‌ها، اندازه دسته، تعداد رئوس، تعداد ویژگی‌ها و ...) را مشخص می‌کنند.

```
adj = get_adjacency_matrix(adj_filename, num_of_vertices,
                           id_filename=id_filename)

adj_dtw = np.array(pd.read_csv(config['adj_dtw_filename'], header=None))

adj_mx = construct_adj_fusion(adj, adj_dtw, 4)
print("The shape of localized adjacency matrix: {}".format(
    adj_mx.shape), flush=True)
```

این قسمت، ماتریس‌های مجاورت را بارگذاری و ترکیب می‌کند.

```
data = mx.sym.var("data")
label = mx.sym.var("label")
adj = mx.sym.Variable('adj', shape=adj_mx.shape,
                      init=mx.init.Constant(value=adj_mx.tolist()))
adj = mx.sym.BlockGrad(adj)
mask_init_value = mx.init.Constant(value=(adj_mx != 0)
                                   .astype('float32').tolist())
```

این قسمت، متغیرهای سمبلیک MXNet را برای داده‌های ورودی، برچسب‌ها، ماتریس مجاورت و مقداردهی اولیه ماسک تعریف می‌کند.

```
filters = config['filters']
first_layer_embedding_size = config['first_layer_embedding_size']
if first_layer_embedding_size:
    data = mx.sym.Activation(
        mx.sym.FullyConnected(
            data,
            flatten=False,
            num_hidden=first_layer_embedding_size
        ),
        act_type='relu'
    )
else:
    first_layer_embedding_size = num_of_features
```

این قسمت، یک لایه جاسازی اولیه (اختیاری) را به داده‌های ورودی اضافه می‌کند.

```
net = stsgcn(
    data, adj, label,
    points_per_hour, num_of_vertices, first_layer_embedding_size,
    filters, module_type, act_type,
    use_mask, mask_init_value, temporal_emb, spatial_emb,
    prefix="", rho=1, predict_length=12
)
```

تابع stsgcn (که قبلاً به طور کامل توضیح داده شد) را فراخوانی می‌کند تا مدل STSGCN را بسازد .تمام پارامترهای لازم (از جمله داده‌های ورودی، ماتریس مجاورت، برچسب‌ها، و پارامترهای مختلف مدل) به stsgcn داده می‌شوند .خروجی stsgcn، که همان مدل ساخته‌شده است، در متغیر net ذخیره می‌شود.

```
assert net.infer_shape(
    data=(batch_size, points_per_hour, num_of_vertices, 1),
    label=(batch_size, num_for_predict, num_of_vertices)
)[1][1] == (batch_size, num_for_predict, num_of_vertices)
return net
```

این خط یک *ادعا* (assertion) است که بررسی می‌کند شکل خروجی مدل (net) با شکل مورد انتظار مطابقت داشته باشد.

به طور خلاصه تابع construct_model تمام مراحل لازم برای ساخت مدل STSGCN را انجام می‌دهد: بارگذاری داده‌ها و تنظیمات، ایجاد متغیرهای سمبلیک، ساخت ماتریس‌های مجاورت، ایجاد یک لایه جاسازی اولیه (اختیاری)، فراخوانی تابع stsgcn برای ساخت هسته اصلی مدل، بررسی شکل خروجی، و در نهایت، برگرداندن مدل ساخته‌شده. این تابع، یک تابع سطح بالا است که کل فرآیند ساخت مدل را مدیریت می‌کند.

```python
def get_adjacency_matrix(distance_df_filename, num_of_vertices,
                         type_='connectivity', id_filename=None):

    import csv
    A = np.zeros((int(num_of_vertices), int(num_of_vertices)),
                 dtype=np.float32)

    if id_filename:
        with open(id_filename, 'r') as f:
            id_dict = {int(i): idx
                       for idx, i in enumerate(f.read().strip().split('\n'))}
        with open(distance_df_filename, 'r') as f:
            f.readline()
            reader = csv.reader(f)
            for row in reader:
                if len(row) != 3:
                    continue
                i, j, distance = int(row[0]), int(row[1]), float(row[2])
                A[id_dict[i], id_dict[j]] = 1
                A[id_dict[j], id_dict[i]] = 1
        return A

    with open(distance_df_filename, 'r') as f:
        f.readline()
        reader = csv.reader(f)
        for row in reader:
            if len(row) != 3:
                continue
            i, j, distance = int(row[0]), int(row[1]), float(row[2])
            if type_ == 'connectivity':
                A[i, j] = 1
                A[j, i] = 1
            elif type == 'distance':
                A[i, j] = 1 / distance
                A[j, i] = 1 / distance
            else:
                raise ValueError("type_ error, must be "
                                 "connectivity or distance!")
    return A
```

تابع get_adjacency_matrix یک فایل CSV حاوی اطلاعات یال‌های گراف را می‌خواند و بر اساس آن، یک ماتریس مجاورت می‌سازد. این تابع می‌تواند دو نوع ماتریس مجاورت ایجاد کند 'connectivity' (باینری) و 'distance' (معکوس فاصله) همچنین، می‌تواند از یک فایل شناسه گره‌ها (id_filename) برای نگاشت شناسه‌های گره‌ها به اندیس‌های ماتریس استفاده کند.

```python
def construct_adj(A, steps):
    N = len(A)
    adj = np.zeros([N * steps] * 2)

    for i in range(steps):
        adj[i * N: (i + 1) * N, i * N: (i + 1) * N] = A

    for i in range(N):
        for k in range(steps - 1):
            adj[k * N + i, (k + 1) * N + i] = 1
            adj[(k + 1) * N + i, k * N + i] = 1

    for i in range(len(adj)):
        adj[i, i] = 1

    return adj
```

تابع construct_adj یک ماتریس مجاورت بزرگتر را با تکرار ماتریس مجاورت کوچکتر A در بلوک‌های قطری و سپس اضافه کردن اتصالات بین گره‌های متناظر در بلوک‌های متوالی و در نهایت اضافه کردن خود حلقه‌ها ایجاد می‌کند این تابع معمولا برای ایجاد یک ماتریس مجاورت برای شبکه‌های عصبی گرافی که روی چندین گام زمانی عمل می‌کنند، استفاده می‌شود (جایی که steps تعداد گام‌های زمانی را نشان می‌دهد.

```python
def construct_adj_fusion(A, A_dtw, steps):

    N = len(A)
    adj = np.zeros([N * steps] * 2)

    for i in range(steps):
        if (i == 1) or (i == 2):
            adj[i * N: (i + 1) * N, i * N: (i + 1) * N] = A
        else:
            adj[i * N: (i + 1) * N, i * N: (i + 1) * N] = A_dtw

    for i in range(N):
        for k in range(steps - 1):
            adj[k * N + i, (k + 1) * N + i] = 1
            adj[(k + 1) * N + i, k * N + i] = 1

    adj[3 * N: 4 * N, 0:  N] = A_dtw
    adj[0 : N, 3 * N: 4 * N] = A_dtw

    adj[2 * N: 3 * N, 0 : N] = adj[0 * N : 1 * N, 1 * N : 2 * N]
    adj[0 : N, 2 * N: 3 * N] = adj[0 * N : 1 * N, 1 * N : 2 * N]
    adj[1 * N: 2 * N, 3 * N: 4 * N] = adj[0 * N : 1 * N, 1 * N : 2 * N]
    adj[3 * N: 4 * N, 1 * N: 2 * N] = adj[0 * N : 1 * N, 1 * N : 2 * N]


    for i in range(len(adj)):
        adj[i, i] = 1

    return adj
```

تابع construct_adj_fusion یک ماتریس مجاورت ترکیبی ایجاد می‌کند که هم اطلاعات ساختار گراف از ماتریس A و هم اطلاعات شباهت سری‌های زمانی از ماتریس A_dtw را در بر می‌گیرد. این ماتریس به گونه‌ای ساخته می‌شود که بخش‌های

زمانی مختلف سری‌های زمانی (که با steps و بلوک‌بندی ماتریس مشخص می‌شوند) بتوانند از طریق اتصالات مبتنی بر DTW با یکدیگر تعامل داشته باشند.

```python
def generate_from_train_val_test(data, transformer):
    mean = None
    std = None
    for key in ('train', 'val', 'test'):
        x, y = generate_seq(data[key], 12, 12)
        if transformer:
            x = transformer(x)
            y = transformer(y)
        if mean is None:
            mean = x.mean()
        if std is None:
            std = x.std()
        yield (x - mean) / std, y
```

تابع generate_from_train_val_test یک مولد (generator) پایتون است که داده‌ها را از یک دیکشنری که شامل کلیدهای 'train', 'val' و 'test' است، بارگذاری می‌کند، آن‌ها را به صورت دنباله‌هایی (sequences) با طول مشخص در می‌آورد، یک تبدیل (transformation) اختیاری روی آن‌ها اعمال می‌کند (مثلاً نرمال‌سازی)، و در نهایت، داده‌های تبدیل‌شده را به صورت دسته‌هایی (batches) تولید (yield) می‌کند.

```python
def generate_from_data(data, length, transformer):
    mean = None
    std = None
    train_line, val_line = int(length * 0.6), int(length * 0.8)
    for line1, line2 in ((0, train_line),
                         (train_line, val_line),
                         (val_line, length)):
        x, y = generate_seq(data['data'][line1: line2], 12, 12)
        if transformer:
            x = transformer(x)
            y = transformer(y)
        if mean is None:
            mean = x.mean()
        if std is None:
            std = x.std()
        yield (x - mean) / std, y
```

تابع generate_from_data بسیار شبیه به تابع generate_from_train_val_test است، اما به جای اینکه داده‌ها را از یک دیکشنری با کلیدهای 'train', 'val' و 'test' بخواند، داده‌ها را از یک آرایه NumPy (یا ساختار مشابه) و با استفاده از تقسیم‌بندی دستی به سه بخش آموزش، اعتبارسنجی و آزمایش، می‌خواند.

```python
def generate_data(graph_signal_matrix_filename, transformer=None):

    data = np.load(graph_signal_matrix_filename)
    keys = data.keys()
    if 'train' in keys and 'val' in keys and 'test' in keys:
        for i in generate_from_train_val_test(data, transformer):
            yield i
    elif 'data' in keys:
        length = data['data'].shape[0]
        for i in generate_from_data(data, length, transformer):
            yield i
    else:
        raise KeyError("neither data nor train, val, test is in the data")
```

تابع generate_data یک مولد (generator) سطح بالاتر است که وظیفه‌ی بارگذاری داده‌های سری زمانی گرافی از یک فایل npz فایل ذخیره‌شده‌ی NumPy و آماده‌سازی آن‌ها برای آموزش مدل را بر عهده دارد. این تابع از دو تابع کمکی generate_from_train_val_test و generate_from_data (که قبلاً توضیح داده شدند) برای تولید داده‌ها به صورت دسته‌بندی‌شده و استانداردسازی‌شده استفاده می‌کند.

```python
def generate_seq(data, train_length, pred_length):
    seq = np.concatenate([np.expand_dims(
        data[i: i + train_length + pred_length], 0)
        for i in range(data.shape[0] - train_length - pred_length + 1)],
        axis=0)[:, :, :, 0: 1]
    return np.split(seq, 2, axis=1)
```

تابع generate_seq داده‌های سری زمانی (یا سری زمانی گرافی) را به دنباله‌هایی (sequences) با طول مشخص برای آموزش و پیش‌بینی تقسیم می‌کند. این تابع، یک آرایه NumPy (یا ساختار مشابه) را به عنوان ورودی می‌گیرد و آن را به دو آرایه NumPy کوچکتر تبدیل می‌کند: یکی برای داده‌های ورودی (آموزش) و دیگری برای داده‌های هدف (پیش‌بینی).

```python
def mask_np(array, null_val):
    if np.isnan(null_val):
        return (~np.isnan(null_val)).astype('float32')
    else:
        return np.not_equal(array, null_val).astype('float32')
```

تابع mask_np یک ماسک بولی (با مقادیر ۰ و ۱) برای یک آرایه NumPy ایجاد می‌کند. این ماسک، عناصری را که با یک مقدار خاص (null_val) متفاوت هستند، مشخص می‌کند. اگر null_val برابر np.nan باشد، ماسک عناصری را که NaN نیستند مشخص می‌کند. این تابع، یک ابزار کمکی برای توابع masked_mape_np, masked_mse_np و masked_mae_np است (که در ادامه توضیح داده می‌شوند)

```python
def masked_mape_np(y_true, y_pred, null_val=np.nan):
    with np.errstate(divide='ignore', invalid='ignore'):
        mask = mask_np(y_true, null_val)
        mask /= mask.mean()
        mape = np.abs((y_pred - y_true) / y_true)
        mape = np.nan_to_num(mask * mape)
        return np.mean(mape) * 100
```

تابع masked_mape_np میانگین درصد خطای مطلق (MAPE) را محاسبه می‌کند، اما با این تفاوت که می‌تواند مقادیر پوچ یا نامعتبر را در داده‌ها نادیده بگیرد. این کار با استفاده از یک ماسک بولی انجام می‌شود. این تابع برای ارزیابی عملکرد مدل‌های پیش‌بینی سری‌های زمانی، به ویژه زمانی که داده‌ها ممکن است شامل مقادیر ازدست‌رفته (missing values) یا پرت باشند، مفید است.

```python
def masked_mse_np(y_true, y_pred, null_val=np.nan):
    mask = mask_np(y_true, null_val)
    mask /= mask.mean()
    mse = (y_true - y_pred) ** 2
    return np.mean(np.nan_to_num(mask * mse))
```

تابع masked_mse_np میانگین مربعات خطا (MSE) را محاسبه می‌کند، اما مقادیر پوچ را در داده‌ها نادیده می‌گیرد. این تابع برای ارزیابی عملکرد مدل‌های پیش‌بینی، به ویژه زمانی که داده‌ها شامل مقادیر ازدست‌رفته یا پرت باشند، مفید است.

```python
def masked_mae_np(y_true, y_pred, null_val=np.nan):
    mask = mask_np(y_true, null_val)
    mask /= mask.mean()
    mae = np.abs(y_true - y_pred)
    return np.mean(np.nan_to_num(mask * mae))
```

تابع masked_mae_np محاسبه‌ی میانگین خطای مطلق (Mean Absolute Error - MAE) را با در نظر گرفتن یک ماسک انجام می‌دهد. این تابع، بسیار شبیه به masked_mse_np است، اما به جای مربع خطا، از قدر مطلق خطا استفاده می‌کند.

۳-۶-آموزش و ارزیابی مدل

در فایل نهایی یعنی main.py مدل را آموزش می دهیم و در آخر نتایج ارزیابی چاپ می شوند.

```python
if isinstance(config['ctx'], list):
    ctx = [mx.gpu(i) for i in config['ctx']]
elif isinstance(config['ctx'], int):
    ctx = mx.gpu(config['ctx'])
```

در این بخش از کد مشخص می کنیم محاسبات روی GPU انجام شود یا CPU

```python
def training(epochs):
    global global_epoch
    lowest_val_loss = 1e6
    for _ in range(epochs):
        t = time.time()
        info = [global_epoch]
        train_loader.reset()
        metric.reset()
        for idx, databatch in enumerate(train_loader):
            mod.forward_backward(databatch)
            mod.update_metric(metric, databatch.label)
            mod.update()
        metric_values = dict(zip(*metric.get()))

        print('training: Epoch: %s, RMSE: %.2f, MAE: %.2f, time: %.2f s' % (
            global_epoch, metric_values['rmse'], metric_values['mae'],
            time.time() - t), flush=True)
        info.append(metric_values['mae'])

        val_loader.reset()
        prediction = mod.predict(val_loader)[1].asnumpy()
        loss = masked_mae_np(val_y, prediction, 0)
        print('validation: Epoch: %s, loss: %.2f, time: %.2f s' % (
            global_epoch, loss, time.time() - t), flush=True)
        info.append(loss)

        if loss < lowest_val_loss:

            test_loader.reset()
            prediction = mod.predict(test_loader)[1].asnumpy()
            tmp_info = []
            for idx in range(config['num_for_predict']):
                y, x = test_y[:, : idx + 1, :], prediction[:, : idx + 1, :]
                tmp_info.append((
                    masked_mae_np(y, x, 0),
                    masked_mape_np(y, x, 0),
                    masked_mse_np(y, x, 0) ** 0.5
                ))
            mae, mape, rmse = tmp_info[-1]
```

این کد، تابع training(epochs) را نشان می‌دهد که حلقه‌ی آموزش (training loop) اصلی مدل STSGCN را پیاده‌سازی می‌کند .این تابع، داده‌های آموزشی را بارگذاری می‌کند، مدل را آموزش می‌دهد، عملکرد مدل را روی داده‌های اعتبارسنجی (validation) ارزیابی می‌کند، و اگر عملکرد مدل روی داده‌های اعتبارسنجی بهبود یافت، عملکرد آن را روی داده‌های آزمایش (test) نیز ارزیابی می‌کند.

نتایج

مدل را با تنظیمات زیر که در فایل config.json آورده شده است آموزش دادیم.

```
{
  "module_type": "individual",
  "act_type": "GLU",
  "temporal_emb": true,
```

```
  "spatial_emb": true,
  "use_mask": true,
  "first_layer_embedding_size": 64,
  "filters": [
      [64, 64, 64],
      [64, 64, 64],
      [64, 64, 64]
  ],
  "batch_size": 32,
  "optimizer": "adam",
  "learning_rate": 1e-3,
  "epochs": 100,
  "max_update_factor": 1,
  "ctx": 0,
  "adj_filename": "matris_mojaverat_fazayi.csv",
  "id_filename": null,
  "graph_signal_matrix_filename": "PEMS08.npz",
  "num_of_vertices": 170,
  "points_per_hour": 12,
  "num_for_predict": 12,
  "num_of_features": 1,
  "adj_dtw_filename": "adj_PEMS08_001.csv"
}
```


مدل ما ۱۲ استپ زمانی آینده را در ۱۰۰ دور آموزش پیش بینی کرده است که نتایج آن در زیر آمده است:

در زیر نمونه ای خروجی آمده است که مقادیر ارزیابی به ازای هر epoch را نشان می دهد که همانطور که مشخص است هرچه تعداد epoch بالاتر می رود پیش بینی مدل ماهم دقیق تر است و خطای کمتری دارد.

```
training: Epoch: 1, RMSE: 131.83, MAE: 103.84, time: 58.11 s
validation: Epoch: 1, loss: 34.54, time: 65.09 s
test: Epoch: 1, MAE: 33.03, MAPE: 31.48, RMSE: 46.73, time: 74.80s

training: Epoch: 2, RMSE: 42.60, MAE: 28.83, time: 61.73 s
validation: Epoch: 2, loss: 28.91, time: 68.93 s
test: Epoch: 2, MAE: 27.58, MAPE: 17.71, RMSE: 40.92, time: 78.81s

training: Epoch: 3, RMSE: 38.47, MAE: 25.32, time: 63.38 s
validation: Epoch: 3, loss: 27.01, time: 70.77 s
test: Epoch: 3, MAE: 25.85, MAPE: 16.31, RMSE: 38.41, time: 80.74s

training: Epoch: 4, RMSE: 36.71, MAE: 23.91, time: 63.23 s
validation: Epoch: 4, loss: 24.32, time: 70.63 s
test: Epoch: 4, MAE: 23.33, MAPE: 15.93, RMSE: 35.29, time: 80.61s

training: Epoch: 5, RMSE: 35.80, MAE: 23.13, time: 63.36 s
validation: Epoch: 5, loss: 24.16, time: 70.75 s
test: Epoch: 5, MAE: 23.18, MAPE: 15.74, RMSE: 35.12, time: 80.80s

step: 5
training loss: 23.13
validation loss: 24.16
tesing: MAE: 23.18
testing: MAPE: 15.74
testing: RMSE: 35.12
```

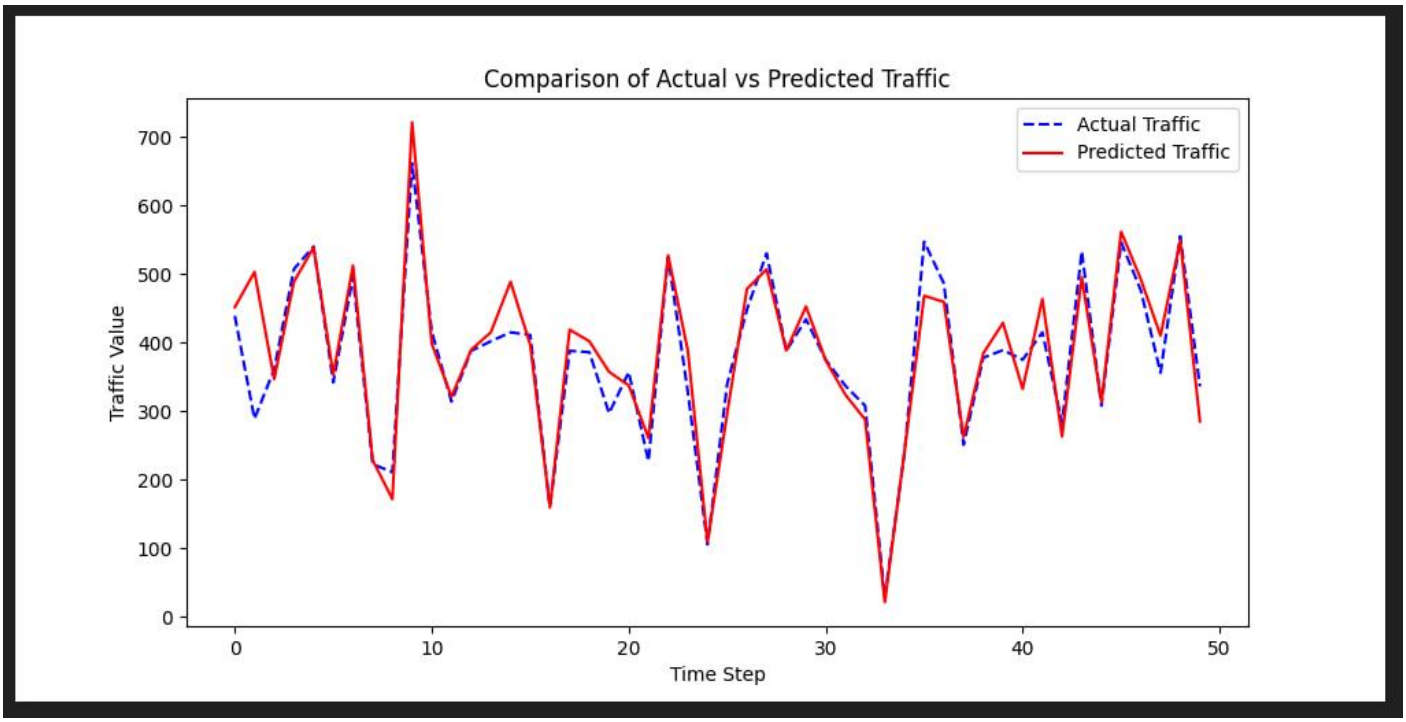
در زیر میانگین معیارهای ارزیابی به ازای هر استپ زمانی آورده شده است.

Step	MAE	MAPE	RMSE
1	14.13270092755219	9.219352875854025	21.742213935253773
2	14.486175779071889	9.412196346901563	22.357161136337172
3	14.784464377734569	9.570536984787559	22.88012865369294
4	15.044804888532145	9.711398561547615	23.34737369552209
5	15.284101048275794	9.840694094400641	23.772715017710695
6	15.515376167320317	9.96518522602245	24.17296140092431
7	15.738315492675385	10.090083395051737	24.55043930190294
8	15.950754622036277	10.20993079431448	24.905296927318364
9	16.15493216859542	10.328629243882698	25.245409137022744
10	16.354894868871174	10.446102849223537	25.576707948479832
11	16.560266710194714	10.565711529051592	25.90856574485762
12	16.783738767764387	10.695564901698592	26.264307226985814

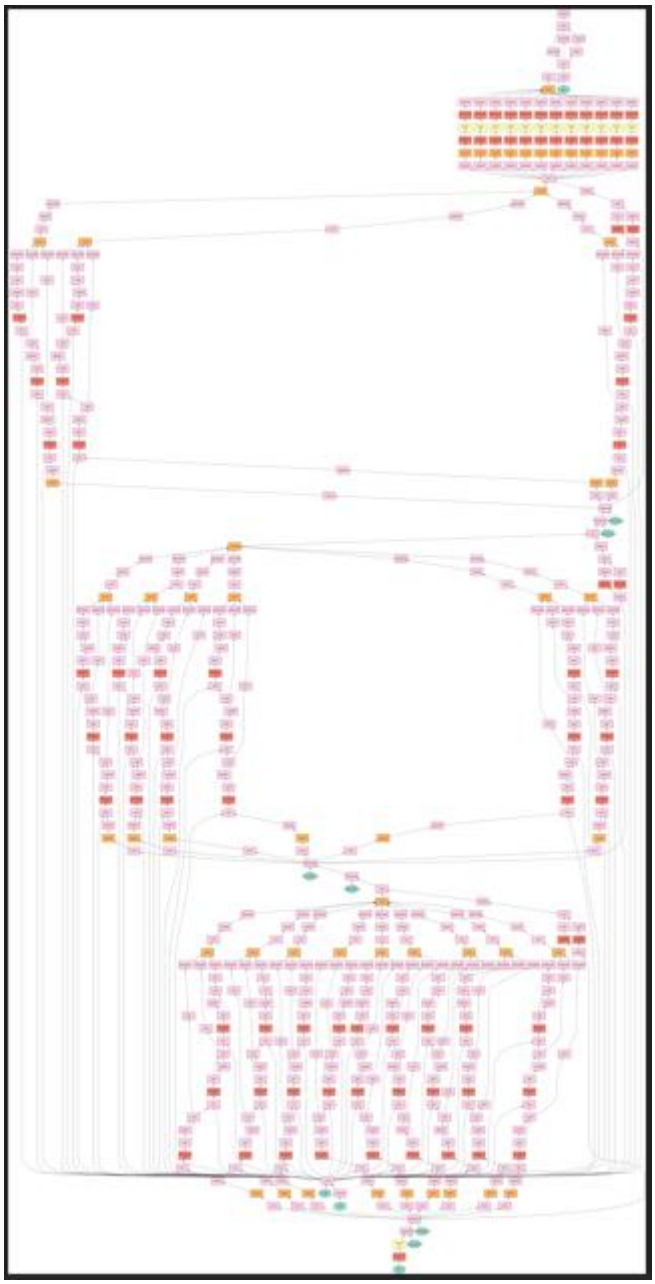
در زیر بخشی از داده های واقعی به همراه مقادیر پیش بینی شده و معیارهای ارزیابی در epoch=100 و گام زمانی آخر آورده شده است.

Step	Data_Point	True_Data	Predicted_Data	MAE	MAPE	RMSE
12	603231	113.0	101.414345	16.783738767848092	10.695564901773613	26.264307226999712
12	603232	193.0	191.29613	16.783738767848092	10.695564901773613	26.264307226999712
12	603233	163.0	181.66393	16.783738767848092	10.695564901773613	26.264307226999712
12	603234	113.0	110.61635	16.783738767848092	10.695564901773613	26.264307226999712
12	603235	76.0	76.62788	16.783738767848092	10.695564901773613	26.264307226999712
12	603236	76.0	72.22033	16.783738767848092	10.695564901773613	26.264307226999712
12	603237	214.0	209.38435	16.783738767848092	10.695564901773613	26.264307226999712
12	603238	162.0	174.13132	16.783738767848092	10.695564901773613	26.264307226999712
12	603239	94.0	114.60783400000001	16.783738767848092	10.695564901773613	26.264307226999712
12	603240	106.0	102.19776	16.783738767848092	10.695564901773613	26.264307226999712
12	603241	152.0	145.9657	16.783738767848092	10.695564901773613	26.264307226999712
12	603242	113.0	114.14558999999998	16.783738767848092	10.695564901773613	26.264307226999712
12	603243	51.0	41.140404	16.783738767848092	10.695564901773613	26.264307226999712
12	603244	113.0	111.577095	16.783738767848092	10.695564901773613	26.264307226999712
12	603245	113.0	111.68087	16.783738767848092	10.695564901773613	26.264307226999712
12	603246	76.0	74.7943	16.783738767848092	10.695564901773613	26.264307226999712
12	603247	127.0	139.75485	16.783738767848092	10.695564901773613	26.264307226999712
12	603248	122.0	101.57624	16.783738767848092	10.695564901773613	26.264307226999712
12	603249	118.0	112.630684	16.783738767848092	10.695564901773613	26.264307226999712
12	603250	76.0	67.20358	16.783738767848092	10.695564901773613	26.264307226999712

در زیر نمودار مربوط به مقادیر واقعی و پیش بینی شده Traffic flow آمده است و همانطور که ملاحظه می شود مقادیر با دقت زیادی پیش بینی شده اند.



شکل گراف تولید شده:



چالش ها

به دلیل کمبود وقت و نبود امکانات کافی نتوانستیم مدل را تعداد دور بیشتری آموزش بدیم و به همان صد دور اکتفا کردیم.

منابع

https://ojs.aaai.org/index.php/AAAI/article/view/16542

https://proceedings.mlr.press/v162/lan22a

https://arxiv.org/abs/2206.09112